Name: Phuc-Hai Huynh. Login: cs61c-cd
Partner: Khanh Dao. Login: cs61c-ef

Question 1:

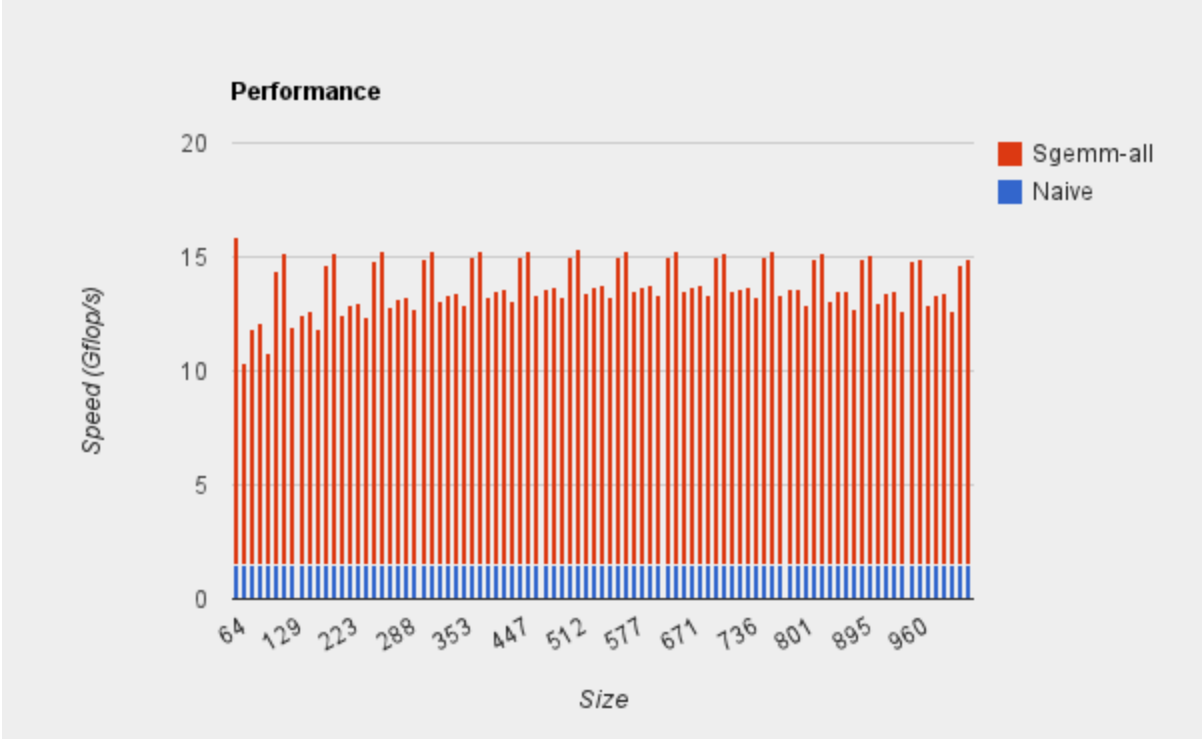
We use part 1 as a function to handle 64x64 matrix multiplications in part 2. We divide the big matrices in 64x64 blocks and do the matrix multiplications for each block. For example, 128x128 C matrix is divided into 4 blocks and then we calculate each block by multiplying the appropriate 64x64 matrix block of A and that of B. We iterate 64x64 matrix blocks in A, B, C sequentially top to down, go right, top to down, go right, and so on. The fringe cases are handled differently to obtain good performance. Fringe cases are 1, 31, 32, 33, 63. For 63, we just use the method explained above and padding 0's to the edge. For 1, we just do the method explained above and calculate the edge by naive method. For 32, we write another 32xn matrix multiplication function to handle fringe cases. For example to get 32x32 matrix in C, we calculate 32xn matrix block in A with nx32 matrix block in B. For 31, handle fringe case by 32xn matrix multiplication function with padding 0's to edge. For 33, handle fringe case by 32xn matrix multiplication function and then calculate the edge naively.

Question 2:

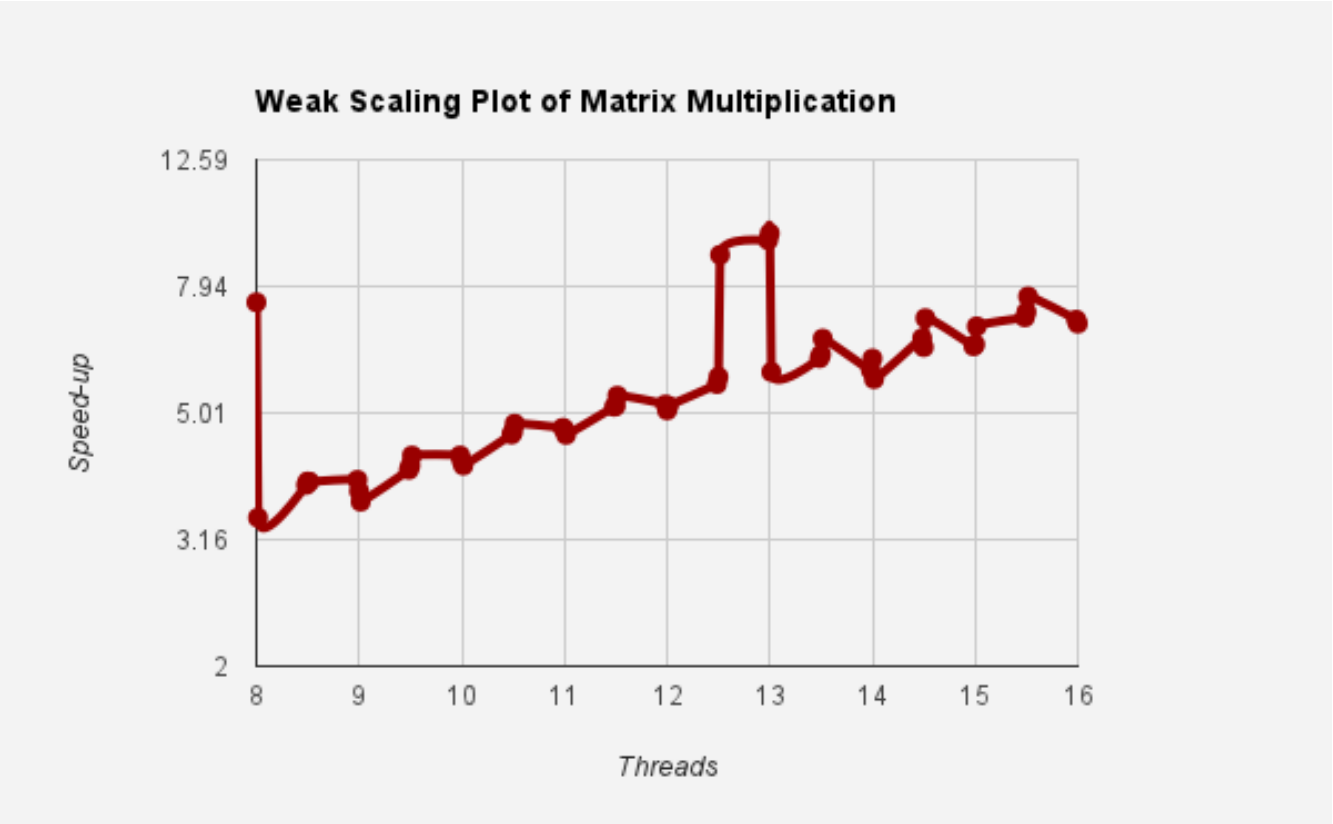
For openmp parallelization, we handle all the fringe cases by 64x64 matrix block multiplications and padding 0's to local buffer if necessary for code organization simplicity. With this code, we just need 2 pragma lines. The first pragma omp parallel line is before the first loop (outer loop). One thing to note that the number of threads is by n / 64 calculations to achieve good performance on medium matrix sizes instead of setting all of the cases to 16. The next line of pragma omp for schedule(dynamic) nowait is also set right before the 1 outer loop to handle the forking of outer for loop.

Question 3:

Speed up of sgemm-all over sgemm-naive



Question 4:
Weak scaling plot



Question 5:
Strong scaling plot